Philipp Drieß, Florian Evers, Markus Brückner

The MoSaKa QoS system: Architecture and evaluation

Original published in:
International Journal on Advances in Telecommunications. - [S.l.] : IARIA. – 5 (2012), 3&4, p. 216-228.
ISSN: 1942-2601
URL: http://www.iariajournals.org/telecommunications/tocv5n34.html
[Visited: 2015-01-28]

The MoSaKa QoS System: Architecture and Evaluation

Philipp Drieß, Florian Evers, and Markus Brückner
Integrated Communication Systems Group
Ilmenau University of Technology, Ilmenau, Germany
Email: {philipp.driess, florian.evers, markus.brueckner}@tu-ilmenau.de

***Abstract*—The provision of Quality-of-Service (QoS) in packet-switched transmissions over highly mobile satellite terminals presents challenges not solved by existing schemes like *Integrated Services* and *Differentiated Services*. Such schemes rely on stable link conditions, a requirement that cannot be guaranteed in a mobile environment. To support robust audio and video conferencing, an end-to-end reservation-based approach is inevitable. This led to the development of the *MoSaKa QoS System*, which combines a reservation-based QoS scheme with the ability to deal with changing link conditions. The main idea was to enable applications to degrade gracefully if an unstable link deteriorates. Each router implements a cross-layer QoS agent, which tracks the network-layer-based QoS and takes the current status of the lower layers into account. Certain flows can be suspended without canceling them if the capacity of a link deteriorates. To select which flow has to be suspended, an *optimizer* was implemented which examines the flows for their priority and respective QoS requirements. To depict how this optimizer works and how the system performs, a testbed with an emulated satellite link was set up. The obtained results show, that the presented system is able to provide appropriate QoS over unstable links.**

Keywords-quality of service; satellite communication; mobile communication; IntServ; signaling

I. Introduction

This paper is an invited paper that is based on a previous work published at the AICT 2012 conference [1]. At that time, no evaluation results were available. These measurement results are the additional contribution of this paper.

First of all, wire-based transmission systems such as Ethernet provide *stable* links. In other words, the transmission conditions such as the link capacity and error rate do not change over time. In contrast, the transmission conditions of wireless systems are considered *unstable*. For example, if a laptop is carried around, it experiences different kinds of fading effects. Similarly, mobile satellite terminals are constantly affected by trees, clouds and other obstacles that impair the line-of-sight transmission to the satellite.

Such links with changing conditions can not be avoided, which leads to problems regarding support for Quality-of-Service (QoS). Reservation-based schemes like *Integrated Services* (IntServ [2]) depend on networks with *stable* links for their capacity management, which fails if the available capacity is a dynamic parameter. In contrast, *Differentiated Services* (DiffServ [3]), an architecture that is based on the differentiation of traffic into classes with specific properties, does not offer reservations at all and thus is not able to offer guarantees to the applications.

Having guarantees might be a requirement, depending on the intended use case. In the research project *Mobile Satellite Communications in Ka-Band* (MoSaKa, see [4] for an introduction), a satellite-based communication system was developed to support rescue teams in disaster scenarios. In such a system, voice communication is one very important application, ideally in combination with video. As satellite resources are scarce, not all communication attempts can be admitted. However, continuous communication streams like voice conversations are not the only kind of traffic in disaster communication systems. A multitude of data has to be exchanged, such as digital maps, status reports, and position information. Therefore, a packet-switched approach that is based on the TCP/IP protocol suite is the most flexible approach to build such a network. To serve important applications like voice reliably, a reservation-based scheme that assures QoS has to be implemented, causing the aforementioned problem regarding the unstable characteristics of the link to the satellite. The availability of a QoS system coping with those limitations will be a key factor for being able to use packet-based satellite communication systems as backbones, especially in disaster scenarios.

In this paper, the MoSaKa QoS system, a novel reservation-based QoS architecture that is able to cope with unstable links, is presented. The focus is on satellite-based networks. MoSaKa empowers rescue teams to communicate in environments without communication infrastructure, which is the research area the MoSaKa project is looking at. However, the algorithms shown in this paper could also be applied to other QoS-enabled wireless transmission systems, such as IEEE 802.11e. The whole QoS system was implemented and is available for the GNU/Linux platform. A testbed was set up, that allowed to perform extensive functional tests and performance evaluations. The results of these tests are presented in this paper.

The remainder of this paper is organized as follows: Section II describes the scenario for which the proposed architecture was designed. The requirements for this architec-

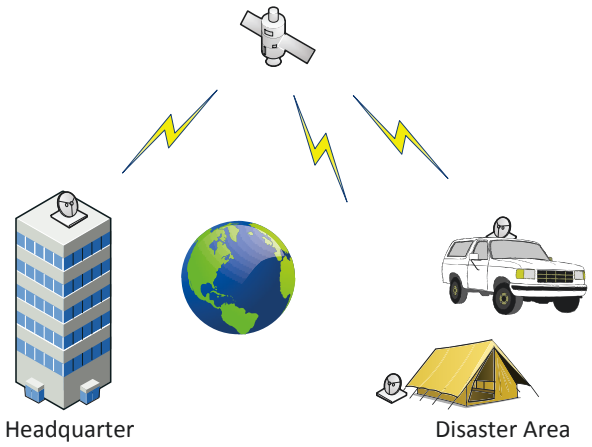

Figure 1. Typical use cases for MoSaKa entities: fixed, nomadic and mobile terminals

ture are derived in Section III. Section IV gives an overview over the related work and highlights the shortcomings of previous approaches. Section V depicts the final architecture of the MoSaKa QoS system by presenting all its functional components. The test environment is presented in Section VI. There the results of multiple functional tests and performance comparisons are provided. The paper is concluded in Section VII with an overview about future work.

II. The MoSaKa Research Project

The MoSaKa project aims at developing a complete satellite communication stack from the antennas up to the QoS management, including antenna tracking systems and a decentralized resource allocation scheme. In this paper, the main focus is on the QoS system as it is seen by the higher layers. Layer 2 and below are only mentioned insofar as they are required to explain design decisions regarding the higher layers.

Figure 1 shows a typical usage scenario: some nomadic and mobile terminals deployed in a remote area use the satellite link to communicate with their headquarter. Each terminal uses a dynamic set of services resulting in individual traffic demands. The communication link from each terminal to the satellite is considered unstable: the link quality fluctuates with the movement of the terminal and changing conditions of the environment. Most of these fluctuations are short, but some may persist for a longer span of time.

Large-scale disaster relief operations cause a significant demand for communication. Satellite links are a – comparatively – scarce resource with only a small capacity and long delays.[1] These two effects have to be considered while implementing a system-wide QoS infrastructure.

[1]For geostationary orbits at a height of $\approx$ 36 000 km the time-of-flight is already longer than 100 ms for one direction.

III. Requirements

Based on the scenario presented in Section II, a set of requirements, that a QoS infrastructure has to fulfill, were derived.

Hard QoS guarantees

Targeted applications, such as for audio and video conferencing, assume a transmission behavior as provided by circuit-switched communication systems. This includes a guaranteed data rate, a deterministic delay, and reasonable low error rate. Providing hard guarantees in IP-based networks can be achieved by end-to-end path reservation schemes. Supporting dynamic reservations requires a signalization scheme, e.g., to setup and cancel paths.

Decentralized resource allocation

A centralized resource management entity is a single point of failure, causes additional signaling traffic towards this entity, and may provide an outdated view of the resources of the whole network especially if the signaling messages are affected by considerable delays. To avoid these effects, a decentralized resource management scheme is preferred.

Efficient handshakes

The main issue in designing an efficient signalization scheme the long transmission delay introduced by the satellite link. With a round trip time of ≈ 400 ms, complex handshakes with multiple messages traveling back and forth are imposing an unacceptable overall delay.

Efficient handling of link instability

Today's QoS systems assume stable links with static resources that can be utilized for reservations. This assumption is no longer valid in mobile, satellite-based communication systems or even in mobile communication systems in general. Over the time, the propagation conditions are subject to change. The QoS infrastructure presented in this paper must be able to cope with such unstable link conditions.

Cross-layer link usage optimization

Satellite-based communication with multiple terminals takes place on a shared broadcast medium. To enable parallel transmissions via one single satellite, the MoSaKa physical and MAC layers have to assign the available link spectrum to all terminals that compete for resources. This happens with respect to the individual resource demands of each terminal. These resource demands are derived from higher-layer QoS requirements that originate from the applications.

Due to the long delay of the broadcast medium, the resource assignment procedure takes place in a distributed manner without central coordination and without any point-to-point negotiation. If the link share of a terminal decreases, the higher layer reservations may not fit into the remaining capacity anymore. In that case, the QoS system has to evaluate

all admitted reservations based on their properties to keep as many of them active as possible.

A resource management system suitable for mobile satellite communication has to address these requirements. Existing solutions fall short in one or the other aspect prompting the development of a new architecture for the MoSaKa project.

IV. Related work

QoS architectures such as IntServ [2], [5], [6] or DiffServ [3], [7] are well known and have a wide range of acceptance. Nevertheless, they have a variety of issues regarding unstable link conditions.

IntServ

IntServ is an architecture that offers hard guarantees regarding QoS parameters. Applications request reservations via a signaling protocol such as the *Resource Reservation Protocol* (RSVP) [8] or *Next Steps in Signaling* (NSIS) [9], [10], [11] to announce their individual traffic requirements. On each node along the transmission path, an IntServ entity manages and monitors the traffic, taking the amount of permitted resources into account.

Applying IntServ upon an unstable link leads to problems if the link capacity decreases. This results in a situation where the sum of all accepted reservations does not fit into the link budget anymore and reservations are violated. As no feedback mechanism is available, the system has to withdraw reservations. Affected applications can only deal with this situation by reserving a new path with different parameters or ceasing communication altogether. Signaling new paths causes additional message load on the already limited link, contributing further to the congestion.

DiffServ

One of the problems of IntServ in large-scale networks is its bad scalability. Due to the state kept in each intermediate node, IntServ installations do not scale to Internet-sized networks. This prompted the development of DiffServ, a system based on differentiation of traffic into classes, which are treated differently by the network. This allows the assignment of transmission priorities to distinguish different types of traffic.

As DiffServ does not support the reservation of a communication path, it does not offer guarantees. Excessive traffic in a single class is able to exceed the link capacity, causing packet loss for all affected applications.

Specialized QoS systems

In addition to the IntServ and DiffServ models, a whole body of research for networks with non-standard conditions exists. Some of these approaches served as an inspiration for a solution to the problem at hand.

INSIGNIA: The INSIGNIA QoS system [12] is designed to work in highly dynamic, mobile ad-hoc networks. The in-band signaling approach provides guaranteed data rates by reserving paths, and is able to adapt fastly to changing conditions of the network. The receiver-based adaption mechanism performs well in highly meshed networks, but is not suited for networks with a single bottleneck in the backbone, such as a satellite link. Furthermore, the reservation model lacks expressiveness, as it does not incorporate QoS parameters like delay and error rate, which are parameters that offer room for optimization in satellite transmission systems.

DARWIN: DARWIN [13] is an approach implementing an IntServ-like model based on a global resource broker. This enables the system to optimize resource utilization among the whole network. Regarding satellite-based communication systems, QoS requests as well as network status notifications towards the broker are affected by high transmission delays, potentially causing instability.

DVB-RCS2: Digital Video Broadcasting; Second Generation DVB Interactive Satellite System (DVB-RCS2) [14] is an ETSI standard to implement an interactive return channel using the standard DVB-S satellite transmission system. The system implements DiffServ as its QoS approach on the layers 2 and 3. Due to the lack of a reservation system, it does not offer hard transmission guarantees.

Inmarsat BGAN: The BGAN service provided by Inmarsat [15] is one of a few high data rate, bidirectional satellite services offering QoS with hard guarantees. Here, access terminals are able to request predefined channels with a fixed bandwidth from the system. Flexible signalization or end-to-end reservations are not possible.

For use in mobile satellite environments, all aforementioned approaches lack certain desirable features. This prompted the development of a new reservation-based approach similar to IntServ. Whereas the scalability of IntServ in large networks is a problem, it is not an issue in the system at hand. The typical satellite communication system for the disaster scenario will contain only a limited number of users. This makes stateful systems in the backbone feasible. Even more, the guaranteed reservation of communication paths as offered by a reservation-based system is crucial for rescue teams in disaster scenarios. MoSaKa aims to create such a reservation-based QoS system and to tackle the challenges arising from unstable links.

V. The QoS architecture of MoSaKa

An IntServ-like architecture such as MoSaKa introduces management entities on each intermediate node as well as on each end system. These entities are aware of all reservations that pass the respective node. In the depicted scenario, static routing in the backhaul is assumed, which ensures that each packet of a flow always takes the same route through the network.

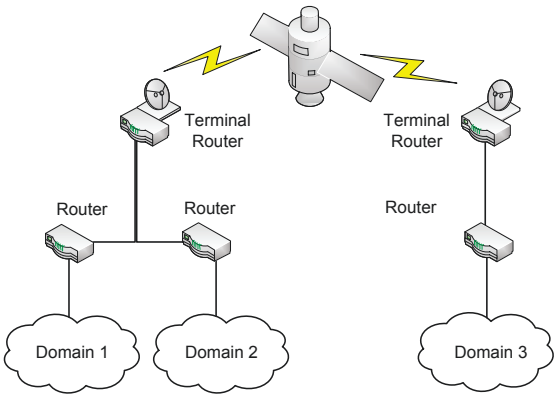

Figure 2. The network of a scenario where the MoSaKa QoS architecture is deployed.

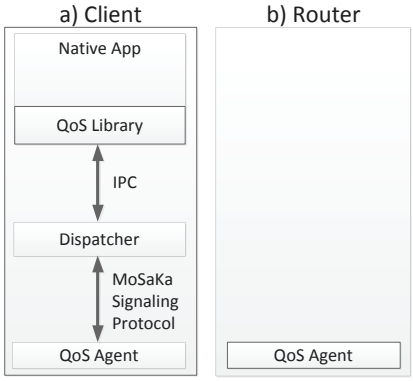

Figure 3. Two kinds of nodes exists in the MoSaKa network: clients and routers. At least, each node requires a QoS Agent running on it. Additional software components are required for clients.

The resulting topology is shown in Figure 2. The central component is the satellite-based communication system with a geostationary satellite and multiple terminals as ground stations. The satellite link is considered to be a bottleneck with a high transmission delay. The terminals act as IP routers and connect local networks to the satellite network. The architecture of the MoSaKa QoS system has no central coordinator.

This approach inherits two issues of IntServ: it has scalability problems and *might* fail if the links are unstable. The former can be neglected with the depicted use case in mind, and the latter is solved by the architecture presented in this paper.

A. Software components

The software components introduced by the MoSaKa QoS architecture are depicted in Figure 3. There are two main components: the QoS Agent and the Dispatcher.

1) The QoS Agent: The QoS Agent is a management entity that exists on each node of the network including routers and clients. This entity is aware of all ongoing reservations that pass the node and has an overview of the transmission resources of each interface that the node possesses. This allows the QoS Agent to decide whether a subsequent reservation can be admitted or has to be rejected. For the purpose of transmitting reservation requests, a signaling protocol such as RSVP or NSIS is required. The QoS Agent intercepts protocol messages and interprets them as necessary. On the satellite terminal, it also communicates with the lower layers to obtain status information regarding the link. This way, it detects link deteriorations.

Furthermore, QoS components like traffic metering and shaping depend highly on the underlying operating system of each node. It is the task of the QoS Agent to adapt the high-level reservations to the QoS primitives available on the node to allow a deployment of the architecture in heterogeneous networks. Each agent consists of a generic part handling the signaling and admission control, and a system-specific part configuring the underlying operating system services.

2) The Dispatcher: The Dispatcher is a component that is only required if a given node has applications running on it, making it a client. A Dispatcher acts as a broker between the applications running on the client and the QoS system in the network. The applications talk to the Dispatcher using *interprocess communication* (IPC). The Dispatcher handles all QoS-related interaction with the network relieving the applications from doing so. Additionally, it serves as an entry point for requests and notifications from the network, decoupling the local application structure and the state saved along the communication path. From the network point of view, the Dispatcher is the entity that holds a reservation and renews it as necessary.

Reservations are always triggered by applications. The Dispatcher merely acts as a proxy. Therefore, applications are part of the MoSaKa QoS architecture and need to be modified to take full advantage of the system. One has to distinguish QoS-enabled applications, legacy applications and translator applications.

QoS-enabled applications: As shown in Figure 4 a, a QoS-enabled architecture includes the MoSaKa QoS library. This library offers an high-level API to interact with the QoS architecture and allows the programmer to request transmission resources or to be notified if an active reservation fails or is suspended due to link deterioration. Such an application is aware of its traffic demands and is able to request the appropriate amount of resources before it starts transmitting. Additionally, it is prepared for incoming feedback messages that indicate that the reservation is currently affected.

Legacy applications: All IP-based applications that exist today are considered as legacy applications. They are not aware of an API to request transmission resources, resulting in traffic that is not known to the QoS infrastructure. Two approaches are possible: this traffic can be considered as *best*

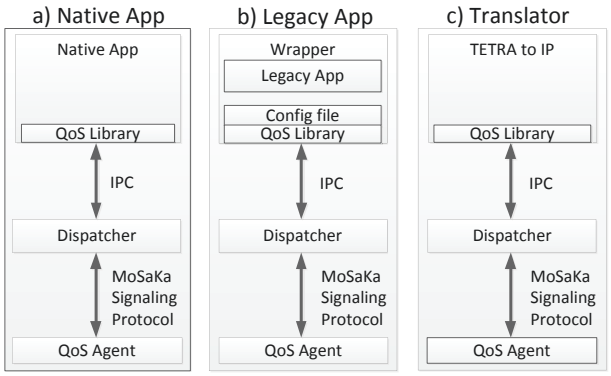

Figure 4. Three kinds of applications are distinguished in the MoSaKa QoS system: Native, legacy and translator applications.

effort traffic, which may or may not pass a bottleneck in the network, or it can be reserved with the help of a *wrapper application* (Figure 4 b).

Such a wrapper application loads a predefined set of QoS requirements from a configuration file, initiates the reservation process and then, if successful, executes the legacy application. In that case, the application does not need to know anything about the QoS system, but benefits from it nevertheless, as the required resources are reserved.

The reservation is held all the time even if the legacy application does not emit any traffic. Even worse, to initiate a reservation, the endpoint must be named, which limits the application to a given set of predefined peers. However, such a wrapper can be seen as an intermediate solution until the affected applications implement the QoS scheme.

Translator applications: Translator applications act as a gateway to other kinds of reservation schemes or networks, e.g. circuit-switched telephony systems. Such an entity is a special case of a QoS-enabled application (Figure 4 c).

In disaster scenarios, connections with other network types such as *Terrestrial Trunked Radio* (TETRA [16]) may be required. A dedicated gateway node with a TETRA base station and a translator application installed on it can interconnect both networks, allowing TETRA terminals to make telephone calls to the headquarter via the satellite. The translator application is aware of the required resources of a TETRA channel, as the traffic requirements of the codecs are known. This allows the translator application to create suitable reservation requests for the MoSaKa network.

B. The QoS-enabled MoSaKa network

The MoSaKa network consists of two kinds of nodes: intermediate nodes are referred to as routers, and end systems are referred to as clients.

As routers have no applications running on them, the only entity required here is the QoS Agent. The routers that are connected to the satellite system are referred to as terminals. On such a terminal the QoS Agent is equipped with additional capabilities to manage the link to the satellite.

Clients, as they are considered as user equipment, have applications running on them. Here, the Dispatcher software is required, to logically connect the QoS-enabled applications with the MoSaKa network.

On each network node the QoS Agent has to configure the local packet forwarding entity of the operating system to stop misbehaving applications from congesting the outgoing interfaces. Above all, it must be impossible that traffic, that exceeds the capacity of the outgoing link, causes packet loss for flows that have been negotiated before. This is achieved by relying on platform specific mechanisms to control traffic flows like *Traffic Control* (tc) and *Netfilter* on Linux. On terminals, this includes parameterizing the MoSaKa MAC scheduler on the data link layer of the satellite link.

C. The signaling scheme

On each client, the QoS-enabled applications communicate with the local Dispatcher via an API offered by the MoSaKa QoS library. By accessing this API, an application informs the QoS System about the amount of resources it requires for a transmission to a well-defined peer. The Dispatcher creates a reservation request signaling message that it sends to its peer entity, the Dispatcher residing on the destination node. All signaling messages are intercepted by each QoS Agent along the path, allowing them to decide whether to accept or to deny this reservation request. If such a reservation has to be denied because of insufficient remaining link resources, a negative acknowledgment is sent back to the initiating Dispatcher. This results in a deletion of the pending reservation on all intermediate nodes and leads to a negative acknowledgment to the application via the API of the QoS Library.

If no node fails along the path, the signaling message reaches the Dispatcher at the destination node. This Dispatcher may be aware of local applications, as they are allowed to register to it beforehand. It offers the opportunity for the applications to modify the incoming request (e.g., to change a data rate of a request to better match the expected traffic). After the local handling is done, the Dispatcher sends back an acknowledgment to the originator. Again, this message is intercepted by all QoS Agents and results in an orderly created reservation along the whole path.

D. Feedback mechanism

To tackle changing link conditions, the signaling protocol was equipped with a feedback mechanism. Feedback messages originate from a QoS Agent in the network, after it detected that one of its observed links deteriorates. This QoS Agent sends a feedback message to all applications that hold reservations affected by this degradation.

On each node, the network interface is monitored by the local QoS Agent. Additionally, it is aware of all active reservations that involve this link. Moreover, it allows to

decide whether the remaining capacity is still high enough to serve all reservations. If the capacity falls below the amount of resources assigned to reservations, the QoS Agent starts to optimize. Optimization is done by building a set of allowed reservations starting from the one with the highest priority. Reservations are incrementally added to this set if there is still capacity available. This simple optimization algorithm is suited well for highly hierarchical communication environments, such as those present during disaster relief operations.

After optimization, the QoS Agent has a list of reservations that still have enough link resources. In addition, it has a list of reservations that do not fit into the link anymore. Instead of canceling these reservations, as architectures such as IntServ would have to do, the QoS Agent of MoSaKa is able to set affected reservations *on hold*. Such a suspended reservation is still known to the whole path, but without any active transmission guarantees. Later, if the link recovers, the reservation is resumed by another feedback message.

In the MoSaKa scenario, it is assumed that link degradations are short in nature. Hence, this approach allows a reservation scheme with a low amount of signaling messages. Applications do not need to actively poll the network for free resources, which otherwise would cause an additional signaling load. Furthermore, the data transmission is resumed faster if the network sends resume messages. Such a message can be sent immediately after the link recovered, instead of relying on the applications that would have to poll the network for a new reservation. Needless to say, even if such a subsequent reservation request would be granted, it would be necessary to complete the full handshaking sequence.

Nevertheless, if the link stays degraded for a longer span of time, the QoS Agent may cancel reservations to prevent congesting the network with reservations that cannot be served anyway.

In the MoSaKa QoS architecture, signaling messages are usually exchanged between Dispatchers on two peer nodes, and are intercepted by all QoS Agents on each intermediate node including the end nodes that run the Dispatchers. If a given reservation has to be suspended, the QoS Agent creates signaling messages and sends them to both Dispatchers, allowing all QoS Agents on the path to notice that this reservation is currently *on hold*.

If a signaling message arrives at a Dispatcher, it passes it to the respective application. Here, it triggers a trap in the QoS library informing the application about an accepted, suspended, resumed or canceled reservation.

This feedback scheme is new and allows a graceful degradation of communication. To underline this, one of the most important applications of the MoSaKa scenario is analyzed: Video chat.

E. Impact on Video chat

If a user starts a video chat session with the headquarter, the video chat application tries to reserve resources for the video data and for the audio data separately. As a video chat session is bidirectional, the reservation message contains resource requests for both directions at the same time. For this reason, the signaling handshake completes fully after just one round trip. If the reservation handshake is completed without a rejection from intermediate systems, the path is considered active and can be utilized by both peers.

If the satellite link deteriorates, this is noticed by the QoS Agents on the satellite terminals. Without sufficient transmission resources, they start the optimization process that results in the less important video streams to be set *on hold* to keep the audio streams active. A feedback message is sent to the Dispatchers at both ends of the path, resulting in the deactivation of the video stream in the application. If the link recovers, another feedback allows the video stream to be resumed. This allows applications to provide an indication based on the network state, increasing user satisfaction by making the process transparent. Otherwise, if the link fails to recover, the reservation is canceled by the network.

The *on hold* state enables the system to bridge short link degradations. Degradations are common in mobile satellite communication systems, and dealing with such situations without canceling paths avoids significant signaling effort. More important, the feedback mechanism allows applications to intelligently react to those changes in the network. Especially for satellite links with long delays and a low capacity, such a scheme is essential.

F. MoSaKa Satellite Terminals

To check whether all active reservations fit into the current link capacity, the QoS Agent has to obtain information regarding the link. For that purpose, technology-dependent functionality is required to interact with Ethernet, IEEE 802.11e or the MAC and PHY layers of the MoSaKa satellite terminals.

The MoSaKa satellite link offers QoS-enabled lower layers. From the point of view of the physical layer the satellite link is a shared medium. Each terminal can be received by every other terminal via the satellite. Therefore, it is necessary to allocate and assign parts of the link spectrum to specific sender terminals to prevent collisions. To accommodate changing link conditions, this allocation is not static but takes place every 250 ms. Each active terminal is assigned a short time slot on the lower layer (L2) signaling channel. In this time slot, it broadcasts its resource request to all other terminals. Based on this information, each terminal applies the same resource assignment procedure and comes up with the same resource allocation vector for the next 250 ms data transmission frame. A reservation on the lower layers is valid for only one slot, and has to be renewed continuously by using the L2 signaling channel.

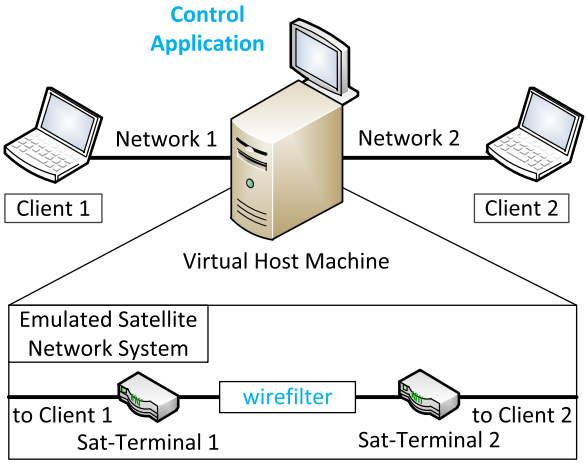

Figure 5. The MoSaKa evaluation testbed comprises two client machines and two virtual satellite terminals connected by an emulated satellite link.

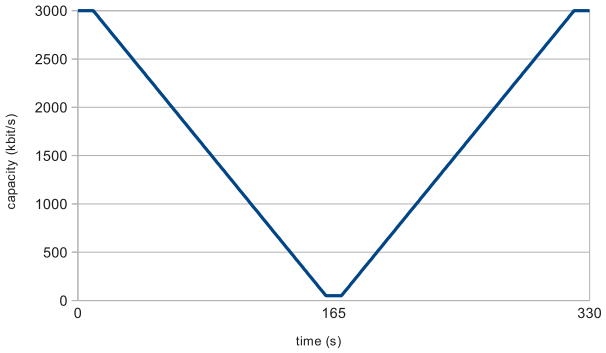

Figure 6. The capacity of the emulated satellite link over time

If the link between one terminal and the satellite deteriorates, the QoS Agent on this terminal gets informed about a lower amount of transmission resources that this terminal got assigned for the next data transmission frame. This allows the QoS Agent to check whether all active high-level reservations still fit into the link share, and to take appropriate actions if they do not.

VI. Evaluation

For the purpose of evaluation, the MoSaKa QoS system was implemented and the testbed depicted in Figure 5 was set up. It incorporates two laptops that act as clients with traffic sources and sinks. A central server is hosting the emulated satellite network components, consisting of two virtualized satellite terminals and an emulated satellite link. For the virtualization, *Kernel-based Virtual Machine* based on Qemu (Qemu/KVM [17]) was chosen as a hypervisor, as it supports hardware virtualization promising performance gains. The host system as well as all guests run a Linux-based operating system with a 3.2.21 kernel. The two guest systems implement the MoSaKa terminal functionality. In addition to the Qemu/KVM hypervisor, Virtual Square *wirefilter* and the *VDE-Switch* [18] are used to connect the guest systems. These components emulate the latency and capacity of a geostationary satellite link. The link parameters are controlled with predefined profiles to emulate arbitrary scenarios in reproducible way.

The measurements were taken using traffic generators adapted to the specific needs of the project. The traffic sources interact with the MoSaKa system to reserve paths. The traffic sinks are able to report received data rate, packet loss rate and transmission delay.

MoSaKa implements a cross-layer approach which tightly integrates the functionalities provided by the layers 1 to 3, whereas the lower layers had to be emulated to some extend. Each satellite terminal runs a MAC layer emulation which controls the forwarding of packets on the VDE-Switch link. This way, a real system behavior is emulated as closely as possible.

A. Emulation of changing link conditions

The testbed runs predefined profiles to make reproducible tests possible. The emulated channel adds a fixed delay of 250 ms, but has a variable capacity between 50 kbit/s and 3000 kbit/s.

Figure 6 shows the capacity curve that was used throughout all tests. It starts with a maximum link capacity of 3000 kbit/s. After 10 s the link starts deteriorating linearly and reaches a minimum capacity of 50 kbit/s at 160 s. At 170 s the link starts to recover linearly and reaches its maximum again at after another 150 s. This basic scenario is sufficient to test the behavior of QoS systems. It could be enhanced to model the behavior of satellite links in a more realistic way, e.g., by incorporating weather conditions. However, this would make it harder to highlight the functionality of the MoSaKa QoS system. Thus, we postponed this for future tests, which will also incorporate transmissions over a real satellite system.

B. Functional tests

Two functional tests were performed to show that the basic QoS functions of the MoSaKa QoS system were working correctly. They reflect the goals of the MoSaKa QoS system, especially:

- The QoS system is dealing with changing link conditions.
- Best effort traffic and reserved traffic are isolated.
- Different priorities are distinguished.
- The performance of UDP and TCP is usable.

1) UDP-only test: In the first test, three independent UDP flows were sent over the emulated satellite link. All were directed in the same direction, to render the satellite link

Table I
QoS REQUIREMENTS FOR ALL PATHS OF THE UDP-ONLY TEST

	reservation@source		emission@source	
flow	priority	data rate	data rate	packet rate
UDP 1	low	849.6 kBit/s	100 kbyte/s	100 p/s
UDP 2	high	1,699.2 kBit/s	200 kbyte/s	200 p/s
Best Effort	(none)	(2,124 kBit/s)	250 kbyte/s	250 p/s

a bottleneck. All UDP packets carried a payload of 1,000 bytes, whereas each packet was extended by an overhead of 62 bytes. This overhead consisted of 8 bytes for the UDPv6 header, 40 bytes for the IPv6 header and 14 bytes for the header and trailer of the Ethernet frame. For an exemplary flow of 100 UDP packets per second with 1000 bytes payload each, the expected data rate on the medium was 849600 bit/s (see Equation 1).

$$100\ \mathrm{p/s} * 1062\ \mathrm{byte/p} * 8\ \mathrm{bit/byte} = 849600\ \mathrm{bit/s} \quad (1)$$

For each flow, table I lists the expected data rate on the medium. This is exactly the data rate that was also reserved for the path, if a reservation for that particular flow was required. Here, the "Best Effort" flow had no reservations, but the flows "UDP 1" and "UDP 2" had requirements regarding their data rate. Furthermore, flow "UDP 2" had a higher data rate and a higher priority than flow "UDP 1".

The MoSaKa QoS system was configured to respect priorities and to use feedback messages to indicate that reservations were suspended or resumed. However, the UDP sources ignored these messages and kept on transmitting UDP packets. As a consequence, if a reservation was suspended, the affected UDP packets were transmitted nevertheless, but as best effort traffic. They then shared the remaining link capacity with all other best effort traffic.

The UDP sources emitted UDP packets of a fixed size. Each packet contained a time stamp and a sequence number, allowing the UDP sinks to calculate the received data rate, the transmission delay and the amount of lost packets.

Measurement results: Figure 7 depicts the sum of the data rates of all three UDP flows at the receiver. It is shown very clear that the added data rate at the receiver matched the capacity of the emulated satellite link (Figure 6). The maximum data rate was at about 350 kbyte/s, which equals – all overhead included – the maximum link capacity of 3000 kbit/s. Furthermore, the figure shows that the reservation for flow "UDP 2" (200 kbyte/s, high priority) was active as long as it fitted into the capacity of satellite link. It was suspended after 75 s, resulting in its UDP packets being transmitted as best effort traffic. This resulted in a decreasing data rate at the receiver, as the traffic was no longer isolated. After 253 s, the satellite link recovered and had enough capacity to resume the reservation for flow "UDP 2". The protection schemes regarding this flow were reactivated, and the reserved data rate was claimed until the test ended.

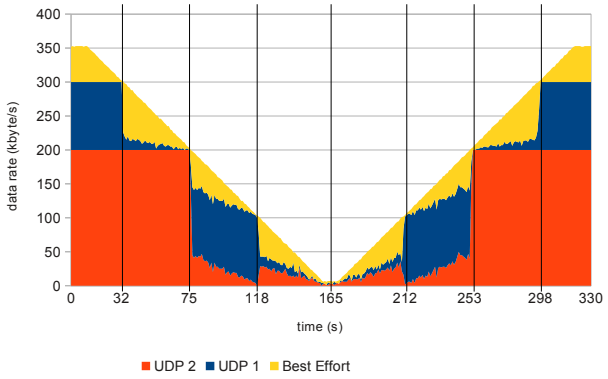

Figure 7. Overall receive data rate during the UDP-only functional test

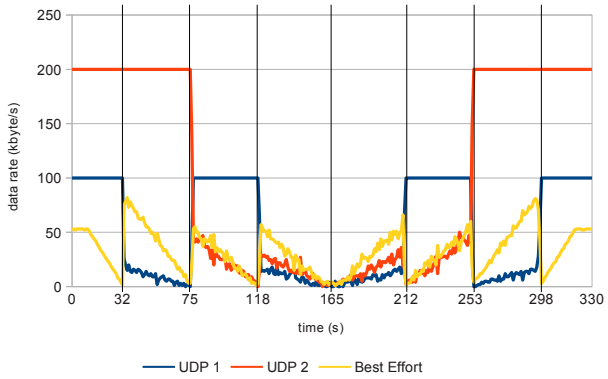

Figure 8. Receive data rate per flow during the UDP-only functional test

Figure 8 visualizes the same set of data, but shows the received data rate of each flow individually. Here, it is very interesting to have a look at flow "UDP 1". This flow was reserved as well, with a lower data rate than flow "UDP 2", but also with a lower priority. As a consequence, when the capacity of the link decreased, flow "UDP 1" was suspended in favor of "UDP 2". This happened the first time at 32 s. Interestingly, after "UDP 2" was suspended at time 75 s, the link had enough capacity to schedule "UDP 1" again, which was resumed immediately. Flow "UDP 1" had protection schemes applied until 118 s, as the capacity became so low that no reservation fitted anymore and only best effort traffic was possible.

During the following phase of link recovery, the same behavior was visible again, with the reservation of "UDP 1" being resumed, suspended, and resumed again. Even if this behavior looks odd, it reflects the expected behavior of a priority-enabled reservation-based QoS system that deals with changing capacities.

As the UDP sinks tracked the sequence number of the

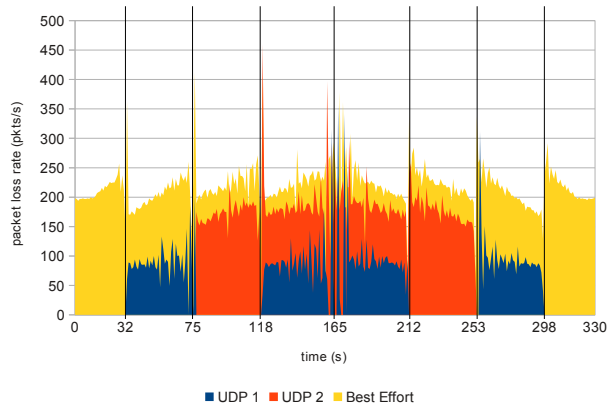

Figure 9. Packet loss rate per UDP flow

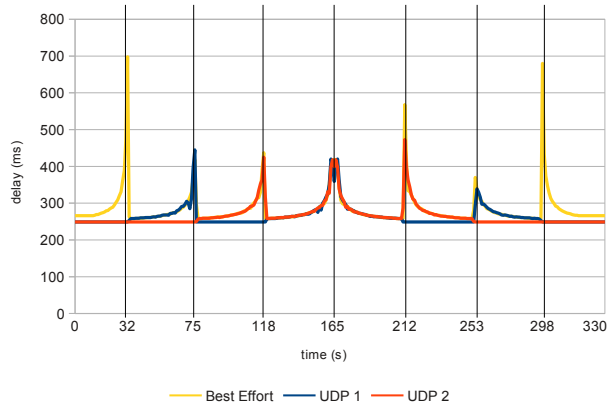

Figure 10. Transmission delay per UDP flow

received packets, they were able to count lost packets. Figure 9 depicts the packet loss rates of all three flows. Flow "UDP 2", which had the highest priority, showed not a single packet loss before 75 s, but underwent a considerable and increasing packet loss afterwards. Flow "UDP 1", that was suspended but resumed later as the link capacity decreased, underwent no packet loss while its reservation was active, but suffered from packet loss while it was suspended. The "Best Effort" flow showed packet loss during the whole test. However, the only purpose of this flow was to create enough traffic to stress the link, in order to check if the isolation schemes regarding reserved flows were working correctly.

For each received UDP packet, the UDP sink calculated the respective transmission delay. All clocks were synchronized and each UDP packet contained a time stamp created by the UDP source. Figure 10 shows the transmission delay of each of the three UDP flows, over the time. UDP packets of flows with an active reservation underwent a transmission delay of ≈ 250 ms, which was equal to the delay emulated by wirefilter. Interestingly, this delay was constant, showing that

Table II
QoS REQUIREMENTS FOR ALL PATHS OF THE TCP/UDP-MIXED TEST

	reservation@source		emission@source	
flow	priority	data rate	data rate	packet rate
TCP 1	low	849.6 kBit/s	not applicable	
TCP 2	high	1,699.2 kBit/s	not applicable	
Best Effort	(none)	(2,124 kBit/s)	250 kbyte/s	250 p/s

the Hierarchical Fair Service Curve (HFSC) packet scheduler of the Linux Kernel did not run out of link capacity. Best Effort traffic, which included traffic of flows that have a suspended reservation, experienced an increasing delay. The increasing delay was caused by buffering, as the packet scheduler had to process a UDP packet rate that did not fit into the capacity of the outgoing link anymore. In addition to that, this effect was also caused by the decreased link capacity itself: decreasing the data rate resulted in an increasing amount of time required to send an Ethernet frame.

2) TCP and UDP coexistence test: In the second functional test, the behavior of TCP (CUBIC) was investigated. A consideration of TCP is very interesting, as the slow start and congestion control functions of TCP are prone to misbehavior if running over high delay links. Furthermore, if TCP and UDP coexist on the same link, UDP is able to block TCP as TCP has intrinsic congestion control functions that UDP does not have.

This test was similar to the first functional test: three flows were sent in the same direction via the satellite link. One flow was a best effort UDP flow carrying 250 kbyte/s, while the other two flows had QoS requirements that were reserved. In terms of data rate and priority, both reservations were similar to the reservations of the UDP-only test. However, the traffic originated from two TCP connections (see table II). Unfortunately, it is challenging to calculate the data rate for the process of reservation, if only the desired data rate regarding the payload of a TCP connection is given. The slow start algorithm of TCP, a variable segment size, and retransmissions due to packet loss are only three examples why TCP might increase the amount of traffic silently.

Instead of calculating the data rate regarding Ethernet, the reservations of the UDP-only test were applied again. As in the first test, the data rates were measured at the socket level, reflecting the *effective* data rate an application can achieve if TCP is used. From the point of view of the applications, the reserved paths should behave like two exclusively assigned long delay paths with a fixed capacity, until the respective reservation is suspended.

Like in the first functional test, the TCP traffic sources ignored feedback messages. Instead, they kept on feeding their socket with user data, even if the respective reservation was suspended. Due to the flow- and congestion control functions of TCP and the limited size of the send buffers in the TCP stack, the applications were throttled if the data could not be transmitted across the network.

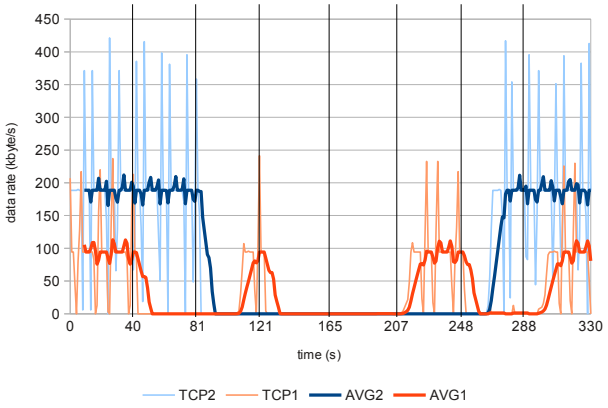

Figure 11. Received data rate per TCP connection

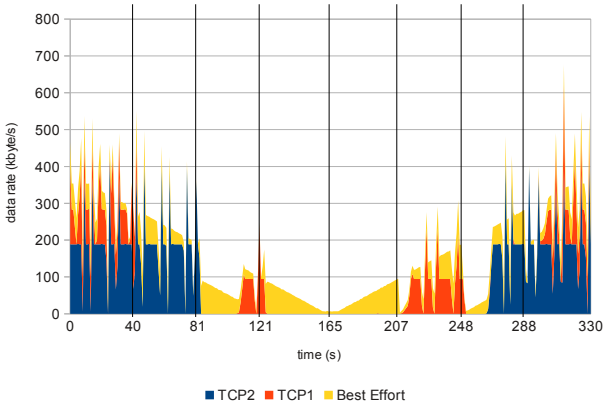

Figure 12. Overall receive data rate of both TCP connections and the best effort UDP flow

Measurement results: Figure 11 shows the data rates of both TCP connections, which reflect the amount of user data read from the TCP sockets at the sink, over time. The best effort UDP traffic is not depicted anymore, as its only purpose was to saturate the emulated satellite link. Each TCP connection is represented by two lines. Lighter colored lines show the received amount of data on a per-second basis, while the darker lines show an averaged versions of the same data, with a sliding window of 10 seconds applied. Filtering was required, as the effective data rates of payload were fluctuating heavily. This was caused by TCP, which guarantees ordered delivery. Received data has to be delayed if gaps exist, which leads to burst of delivered data if these gaps are closed later.

Both TCP connections were able to saturate their paths, as long as their respective reservation were active. At 40 s, the overall link capacity dropped below the sum of both reserved paths, and “TCP 2” was suspended due to its lower priority. The data rate of “TCP 2” dropped to zero, as it had to compete with the traffic of the best effort UDP flow. At 81 s, the link capacity fell below the limit of holding the reservation of “TCP 1”, and its path was suspended too. Thus, the reservation of “TCP 2” fitted again, and its path was resumed. Interestingly, “TCP 2” was not able to immediately reclaim this capacity, but required nearly 30 seconds to start delivering data to the sink application again. This may be due to the intrinsic retransmission timers and the slow start mechanisms of TCP. Future work will have to look deeper into this topic.

At 121 s, the reservation of “TCP 2” was suspended again due to lacking resources, and both TCP connections were blocked by the best effort UDP flow. As the reservation of “TCP 2” was resumed at 207 s, the delivery of data started within a short amount of time. If compared to 81 s, different recovery behaviors of “TCP 2” were observed. The authors suspect, that these differences were caused by synchronization issues regarding the retransmission timers of TCP. Different TCP implementations for satellite systems or unstable links may show an improved performance.

Figure 12 depicts an interesting artifact of the testbed that was caused by the emulated satellite link. Here, the overall data rate of all three flows is shown over time. At 81 s, the reservation of “TCP 1” was suspended and the reservation of “TCP 2” was resumed. As discussed, the data rate of the first connection dropped to zero, while the data rate of the second connection did not recover. Thus, the reservation of “TCP 2” was active, but its resources laid idle. The expected behavior of a QoS system would be that idle resources are assigned to best effort traffic, but Figure 12 shows a different behavior. Here, best effort traffic did not claim idle resources of active reservations, resulting in a fall-off between 81 s and 95 s. Future research will have to look into the behavior of the real MoSaKa layer 2 system.

3) Concluding the functional tests: Both functional tests showed that the MoSaKa QoS system provided end-to-end guarantees. The UDP-only test revealed a perfect match of the reserved and the delivered data rate. Reserved paths did not influence each other and were protected against best effort traffic. When the link capacity decreased, paths were suspended in the order of their priority. As the affected UDP sources did not react to the feedback messages, their respective traffic was treated as best effort traffic.

Due to the perfect isolation of paths, reserved TCP connections were able to utilize the capacity offered by their respective reservation. However, the tests indicated that the path suspension mechanism may collide with TCP’s intrinsic slow start and retransmission timers. TCP connections were not able to take advantage of short intervals of activated reservations. As TCP is the most important transport protocol in IP-based networks, future work should look into possibilities to fully understand and possibly mitigate this issue by either evaluating feedback messages directly inside

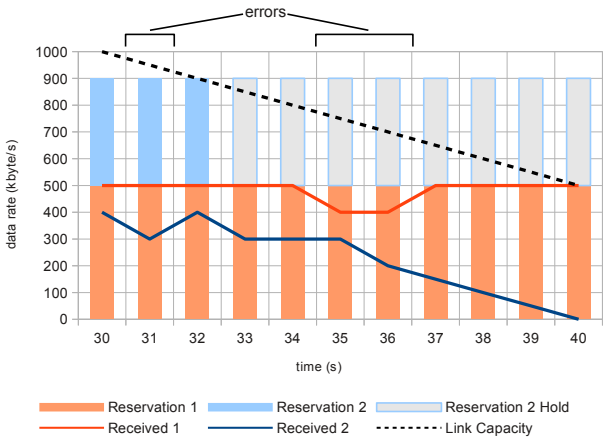

Figure 13. Example of the error analysis according to the MoSaKa QoS system

the TCP protocol entity or by taking network-layer path stability requirements into account.

C. Performance evaluation

In order to evaluate the improvement provided by using the MoSaKa QoS system, it was compared to other methods.

Measurement setup: All comparisons were performed using random sets of 32 reserved UDP-based flows with a total reserved data rate of 3000 kbit/s. Each UDP source emitted a stream of packets with 1000 bytes payload and a constant data rate. The data rate matched the reserved capacity. While the sources involved the MoSaKa QoS system to reserve paths, they did not evaluate any feedback messages. Thus, they did not adapt their output rate in case of a suspended reservation. On the peer node, the UDP sinks reported the received data rate, the packet loss rate and the delay. The link conditions were emulated as described in Section VI-A.

Each test series was evaluated against an independent implementation, checking three rules for each time slot:

1) Paths must be prioritized correctly. A path with a higher priority must never be blocked by one with a lower priority.
2) If a path still fits into the remaining capacity, it must be enabled.
3) Enabled paths must not lose any packets.

Figure 13 illustrates a synthetic example of the analysis process. It comprises two reserved paths, with “Reservation 1” having a high priority and “Reservation 2” having a low priority. Each path has a specific part of the capacity reserved. When the available link capacity falls below the sum of the two reservations at 33 s, “Reservation 2” is suspended. The respective traffic loses its guarantees, and the data rate at the sink falls below its reservation. This is not counted as an error, as this is the specified system behavior in case of a degrading link.

On the other hand, the time slots at 35 s and 36 s are counted as errors regarding “Reservation 1”, and the time slot at 31 s shows an error regarding “Reservation 2”. For each time slot, one can calculate which reservations should be activated and which reservations should be suspended. While flows with a suspended reservation are allowed to undergo packet loss, flows with active reservations must not. If one sink reports packet loss nevertheless, this is counted as an erroneous time slot.

Besides this, the independent implementation of the analysis software does not reason about the source of the errors, but merely implements a black box test with regard to the network. This opens up the opportunity to test against network configurations which support less extensive QoS models than MoSaKa does, or no QoS at all.

Four different QoS models are compared in this paper:

Without QoS: This QoS model served as the baseline representing the current Internet without any QoS guarantees. While individual flows still had an assumed priority and an expected data rate, the network did not obey this information.

This scenario was expected to perform the worst. Usually, routers try to assign available resources fairly, leading to a dropping of packets distributed over all flows.

Without priorities, cancel only: This simple QoS model assumed end-to-end guarantees for reserved flows. The network was able to cancel paths if link conditions deteriorated. However, canceled paths were never resumed. Having different priorities was not supported. This model corresponds to a simple IntServ system that does not offer having priorities.

Even without having priorities and without allowing suspended paths to recover, this model was expected to perform significantly better than the first model. At least for paths that still have an active reservation, the error level should decrease significantly.

With priorities, cancel only: This advanced QoS model allowed having priorities, but it still lacked the ability to re-enable a canceled path after the condition of a link recovered.

Having the ability to obey priorities, such a system should show an even lower error level. The reservations of high priority flows should be kept active longer, leading to less erroneous time slots.

MoSaKa QoS: The MoSaKa QoS model supported both path suspension and obeys priorities. It represents the contribution of the authors.

With both mechanisms in place, MoSaKa should be able to perfectly match the expected behavior, resulting in an error level of zero.

Measurement results: For each of the four QoS models, 20 test series were performed and evaluated, using the same setup as described earlier. The number of erroneous time slots of all 32 flows were averaged, and these averaged error levels were averaged again among all 20 test series. The resulting

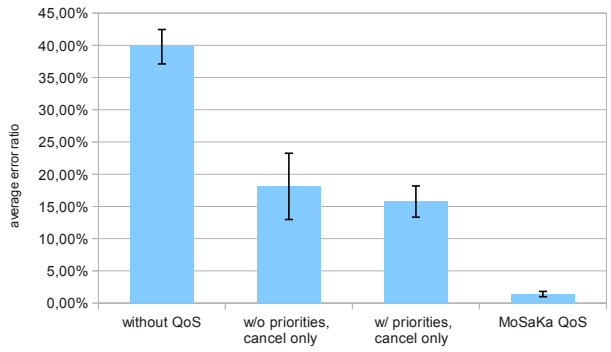

Figure 14. Average error levels of different QoS models

four error levels and their respective standard deviation are shown in Figure 14.

As expected, the first model without any QoS performed worst. If the link capacity dropped below the sum of all flows, the network started dropping packets of all flows. This affected all paths in the same way with no regard to priorities.

Only flows with a low priority showed low error levels, as the analyzer expected these flows to be suspended early. For flows with higher priorities, the error levels increased. On average, in approximately 40% of the time slots of a path's lifetime, the system was not able to provide the *desired* QoS.

Introducing a QoS model, at first a simple one, improved the situation. Even without obeying priorities or providing path suspension, the error levels dropped to 18% on average. If a path was canceled, its resources were freed. These free resources helped keeping the remaining reservations active for a longer time.

Interestingly, testing an advanced QoS model that respects priorities showed only a slight improvement. The average error levels dropped to approximately 16%. With the given link capacity curve as depicted in Figure 6, the most erroneous time slots happened in the second half of each test series. Once a path was canceled, it never recovered. Therefore, its traffic had to compete in the best effort class. This lead to a similar situation as in the first model (without QoS), where packet loss affected all paths uniformly.

Having the full feature set of the MoSaKa QoS system, error level dropped to 1.4%. This was not a perfect result, but most of the errors can be attributed to imperfections in the measurement setup. Having time slots of a given duration leads to false negatives when paths switch state within a slot. Decreasing slot durations would help, but that would lead to other problems: introducing shorter slot times complicates the assessment of flows, especially if they have a low packet rate. Then, many of the slots are empty, and the system starts to detect scheduling effects which might move packets between slots, wrongly leading to errors again.

As expected, MoSaKa performed nearly perfect under the given scenario. Paths were suspended in the correct order and returned to service once the conditions improved.

VII. Conclusion and Future Work

In this paper, a novel QoS architecture called "MoSaKa QoS" was presented. It was designed for networks that involve mobile satellite-based communication links. Previous reservation-based approaches like IntServ are not suited, as the link to the satellite is considered unstable, i.e., provides a varying data rate. In MoSaKa QoS, this problem was circumvented by the introduction of a novel feedback mechanism, that suspends reservations without canceling them. The applications are informed that their reservation was suspended, allowing them to degrade gracefully. To provide an example, a video conferencing software was equipped with these features.

Furthermore, a testbed was set up, containing all components of the MoSaKa QoS system, besides "real" MoSaKa terminals that communicate via a satellite. Instead of a transmission via a satellite, the respective link was emulated with respect to delay and a capacity that changes over time. Extensive measurements were obtained, showing that the reservation-based QoS system of MoSaKa is working correctly. Additionally, it was shown that TCP is able to saturate the path despite the high delay of the link. Apart from that, the performance of TCP went down if no respective reservation was active, and the TCP connection had to share transmission resources with other best effort traffic. Hence, the MoSaKa QoS system enables TCP to be used over congested bottlenecks, even if the end-to-end connection has a high delay.

Besides that, the MoSaKa QoS system was compared to similar approaches. It was shown that the combination of a feedback mechanism and different priorities for reservations offer the best compliance regarding individual QoS requirements. Without feedback, applications have to poll for resources or stop communication altogether. Without distinguishing priorities, the network may suspend flows that are important to the user.

In brief, the MoSaKa QoS system combines the versatility of packet-switched networks with QoS parameters offered by circuit-switched systems. Thus, it allows the coexistence of high-priority voice communication and other low-priority traffic, which is an essential feature of a communication network designed for rescue teams in disaster relief missions.

The components of the MoSaKa QoS system were implemented and are available for GNU/Linux-based systems. Moreover, a testbed for system-wide tests was set up. However, the "real" MoSaKa satellite terminals with their own L2 and L1 components were not operational yet. Thus, as a first step, the satellite link was emulated by using the Qemu/KVM and VDE-tool packages. The results were as expected. Furthermore, it is interesting to check whether the same tests, applied on a testbed involving a

real satellite link, lead to similar results. Besides that, future tests will incorporate sophisticated models regarding terminal movement and weather conditions, providing a more realistic model of the capacity of the emulated satellite link.

From the point of view of the architecture, further research might look into alternative QoS models based on probability distributions instead of hard thresholds. This includes novel reservation models, which provide sophisticated ways of expressing requirements. This enables the system to adapt better to changing link conditions without consulting the applications.

Regarding the network topology, the current MoSaKa system was designed to have a static IPv6-based routing setup. In the future, the system should adapt to various routing protocols, to enable mobility at the network level. In addition, it is beneficial to add support for QoS-enabled multicast traffic, which is currently not supported by our signaling protocol.

Future research should also investigate the possibilities opened up by the MoSaKa feedback mechanism. Currently available audio and video codecs offer various output profiles with different data rates and quality settings. An integration with the MoSaKa QoS system, with the possibility to specify multiple possible data rates per reservation, promises an advanced scheme for graceful degradation.

Moreover, it is not clear yet how the MoSaKa QoS system should interact with non-MoSaKa end systems, such as web servers in the Internet. Currently, as there is no Dispatcher on the peer node, the signaling handshake does not complete. Thus, there is no reservation on the satellite link, rendering TCP unusable if most of the resources are occupied. Possibly, these issues can be solved by introducing translator applications such as proxies, or DiffServ-like classification approaches.

REFERENCES

[1] P. Drieß, F. Evers, and M. Brückner, "A Resource Management Architecture for Mobile Satellite-based Communication Systems," in *The Eighth Advanced International Conference on Telecommunications, AICT 2012*, Stuttgart, 05 2012.

[2] J. Wroclawski, "The Use of RSVP with IETF Integrated Services," *RFC 2210*, September 1997.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," *RFC 2475*, December 1998.

[4] M. Hein, A. Kraus, R. Stephan, C. Volmer, A. Heuberger, E. Eberlein, C. Keip, M. Mehnert, A. Mitschele-Thiel, P. Driess, and T. Volkert, "Perspectives for Mobile Satellite Communications in Ka-Band (MoSaKa)," in *EuCAP 2010: The 4th European Conference on Antennas and Propagation*, Barcelona, Spain, 04 2010.

[5] J. Wroclawski, "Specification of the Controlled-Load Network Element Service," *RFC 2211*, September 1997.

[6] S. Shenker, C. Partridge, and R. Guerin, "Specification of Guaranteed Quality of Services," *RFC 2212*, September 1997.

[7] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," *RFC 2474*, December 1998.

[8] A. Mankin, Ed., F. Baker, B. Braden, S. Bradner, M. O'Dell, A. Romanow, A. Weinrib, and L. Zhang, "Resource ReSerVation Protocol (RSVP) – Version 1 Applicability Statement Some Guidelines on Deployment," *RFC 2208*, September 1997.

[9] R. Hancock, G. Karagiannis, J. Loughney, and S. V. den Bosch, "Next Steps in Signaling (NSIS): Framework," *RFC 4080*, June 2005.

[10] J. Manner, G. Karagiannis, and A. McDonald, "NSIS Signaling Layer Protocol (NSLP) for Quality-of-Service Signaling," *RFC 5974*, October 2010.

[11] E. G. Ash, E. A. Bader, E. C. Kappler, and E. D. Oran, "QSPEC Template for the Quality-of-Service NSIS Signaling Layer Protocol (NSLP)," *RFC 5975*, October 2005.

[12] S.-B. Lee, G.-S. Ahn, X. Zhang, and A. T. Campbell, "INSIGNIA: An IP-Based Quality of Service Framework for Mobile ad Hoc Networks," *Journal of Parallel and Distributed Computing*, no. 60, pp. 374–406, 2000.

[13] P. Chandra, Y.-H. Chu, A. Fisher, G. Jun, C. KosaK, T. E. Ng, P. Steenkiste, E. Takahashi, and H. Zhang, "Darwin: Customizable Resource Management for Value-Added Network Services," *IEEE Network*, vol. 15, no. 1, 2001.

[14] *Digital Video Broadcasting (DVB); Second Generation DVB Interactive Satellite System (DVB-RCS2); Part 3: Higher Layers Satellite Specification*, ETSI, 5 2012, ETSI TS 101 545-3 V1.1.1 (2012-05).

[15] "Inmarsat – BGAN," 12 2012. [Online]. Available: http://www.inmarsat.com/services/BGAN

[16] E. Re, M. Ruggieri, and G. Guidotti, "Integration of TETRA with Satellite Networks: A Contribution to the IMT-A Vision," *Wirel. Pers. Commun.*, vol. 45, pp. 559–568, June 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1363306.1363328

[17] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 41–41. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247360.1247401

[18] R. Davoli, "VDE: Virtual Distributed Ethernet," in *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 213–220. [Online]. Available: http://dl.acm.org/citation.cfm?id=1042447.1043718